分布式选举

2022年2月16日 9:46

why.

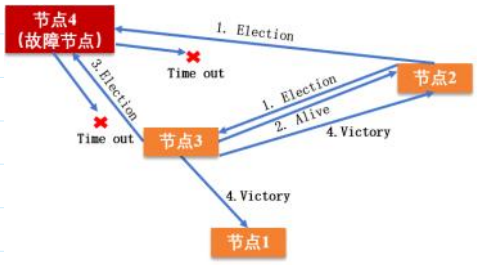
选出一个master node，由它协调和管理其他node，

以保证集群有序运行和节点数据的一致性.

算法

①长者为大：Bully.

- Election 消息，用于发起选举；
- Alive 消息，对 Election 消息的应答；
- Victory 消息，竞选成功的主节点向其他节点发送的宣誓主权的消息。



注：节点i的ID值即为i

Bully 算法选举的原则是“长者为大”，意味着它的**假设条件是，集群中每个节点均知道其他节点的 ID。**在此前提下，其具体的选举过程是：

1. 集群中每个节点判断自己的 ID 是否为当前活着的节点中 ID 最大的，如果是，则直接向其他节点发送 Victory 消息，宣誓自己的主权；
2. 如果自己不是当前活着的节点中 ID 最大的，则向比自己 ID 大的所有节点发送 Election 消息，并等待其他节点的回复；
3. 若在给定的时间范围内，本节点没有收到其他节点回复的 Alive 消息，则认为自己成为主节点，并向其他节点发送 Victory 消息，宣誓自己成为主节点；若接收到来自比自己 ID 大的节点的 Alive 消息，则等待其他节点发送 Victory 消息；
4. 若本节点收到比自己 ID 小的节点发送的 Election 消息，则回复一个 Alive 消息，告知其他节点，我比你大，重新选举。

①优点.

选举快、算法复杂度低、实现简单.

②缺点.

①存储：每个节点有全局的节点信息.

②协调：增减节都可能导致频切换.

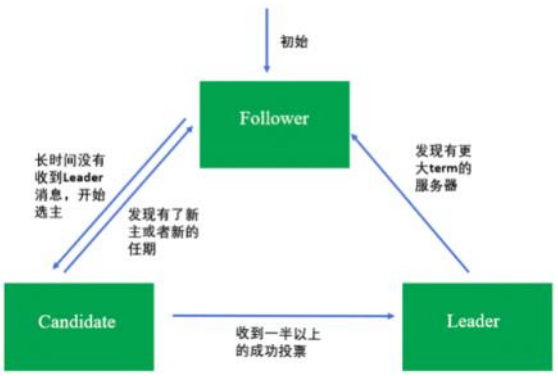
③实例.

Mongo DB.

②民主投票 Raft.

采用 Raft 算法选举，集群节点的角色有 3 种：

- **Leader**，即主节点，同一时刻只有一个 Leader，负责协调和管理其他节点；
- **Candidate**，即候选者，每一个节点都可以成为 Candidate，节点在该角色下才可以被选为新的 Leader；
- **Follower**，Leader 的跟随者，不可以发起选举。



1. 初始化时，所有节点均为 Follower 状态。
2. 开始选主时，所有节点的状态由 Follower 转化为 Candidate，并向其他节点发送选举请求。
3. 其他节点根据接收到的选举请求的先后顺序，回复是否同意成为主。这里需要注意的是，在每一轮选举中，一个节点只能投出一张票。
4. 若发起选举请求的节点获得超过一半的投票，则成为主节点，其状态转化为 Leader，其他节点的状态则由 Candidate 降为 Follower。Leader 节点与 Follower 节点之间会定期发送心跳包，以检测主节点是否活着。
5. 当 Leader 节点的任期到了，即发现其他服务器开始下一轮选主周期时，Leader 节点的状态由 Leader 降级为 Follower，进入新一轮选主。

①优点.

选举快、算法复杂度低、实现简单.

稳定性比Bully好.

稳定性比Bully好.

② 缺点

要求每个节点间都可以互相通信.

③ 实例.

k8s采用的etcd组件.

③ 具有优先级的民主投票：ZAB.

使用 ZAB 算法选举时，集群中每个节点拥有 3 种角色：

- **Leader**，主节点；
- **Follower**，跟随者节点；
- **Observer**，观察者，无投票权。

选举过程中，集群中的节点拥有 4 个状态：

- **Looking 状态**，即选举状态。当节点处于该状态时，它会认为当前集群中没有 Leader，因此自己进入选举状态。
- **Leading 状态**，即领导者状态，表示已经选出主，且当前节点为 Leader。
- **Following 状态**，即跟随者状态，集群中已经选出主后，其他非主节点状态更新为 Following，表示对 Leader 的追随。
- **Observing 状态**，即观察者状态，表示当前节点为 Observer，持观望态度，没有投票权和选举权。

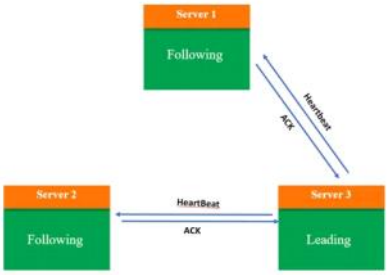
投票过程中，每个节点都有一个唯一的三元组 (server_id, server_zxID, epoch)，其中 server_id 表示本节点的唯一 ID；server_zxID 表示本节点存放的数据 ID，数据 ID 越大表示数据越新，选举权重越大；epoch 表示当前选取轮数，一般用逻辑时钟表示。

ZAB 选举算法的核心是 “少数服从多数，ID 大的节点优先成为主”，因此选举过程中通过 (vote_id, vote_zxID) 来表明投票给哪个节点，其中 vote_id 表示被投票节点的 ID，vote_zxID 表示被投票节点的服务器 zxID。**ZAB 算法选主的原则是：server_zxID 最大者成为 Leader；若 server_zxID 相同，则 server_id 最大者成为 Leader。**

第一步：当系统刚启动时，3 个服务器当前投票均为第一轮投票，即 epoch=1，且 zxID 均为 0。此时每个服务器都推选自己，并将选票信息 <epoch, vote_id, vote_zxID> 广播出去。

第二步：根据判断规则，由于 3 个 Server 的 epoch、zxID 都相同，因此比较 server_id，较大者即为推选对象，因此 Server 1 和 Server 2 将 vote_id 改为 3，更新自己的投票箱并重新广播自己的投票。

第三步：此时系统内所有服务器都推选了 Server 3，因此 Server 3 当选 Leader，处于 Leading 状态，向其他服务器发送心跳包并维护连接：Server1 和 Server2 处于 Following 状态。



① 优点

性能高，对系统无影响.

② 缺点

广播风暴.

选举时间相对较长：增加了对此节点ID和数据ID.

为什么"多数派选举"采用奇数节点？

方便选择.

选举和一致性的关系

选举就是一致性的协商，写到 leader，同步到follower才能更好保证一致性.

集群存在"双主"

关于双主的情况，一般是因为网络故障，比如网络分区等导致的。出现双主的期间，如果双主均提供服务，可能会导致集群中数据不一致。所以，需要根据业务对数据不一致的容忍度来决定是否允许双主情况下提供服务。

	Bully算法	Raft算法	ZAB算法
选举消息回复类型	alive消息	同意或不同意选举的消息	投票信息 <epoch, vote_id, vote_zxID>
Leader选举机制	偏向于让ID更大的节点作为Leader	收到过半数的投票，则当选为Leader	倾向于让数据最新或者ID值最大的节点作为Leader
选举过程	只要节点发现Leader无响应时，或者ID较大的节点恢复故障时，就会发起选举	每个角色为Candidate的节点可参与竞选Leader，且每一个Follower只有一次投票权，即同意或者不同意Candidate的选举	每个节点都可以处于Looking状态参与竞选，都可以多次重新投票，根据epoch、zxID、server_id来选择最佳的节点作为Leader
选举所需时间	短	较短	较长
性能	Bully < Raft < ZAB